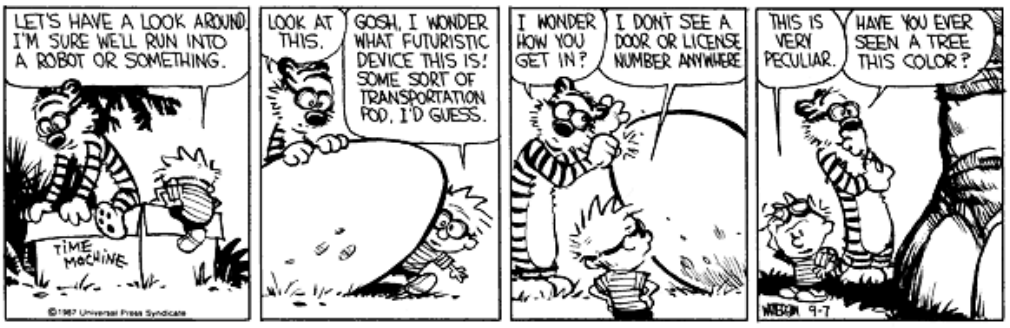
Algorithmique
Contrôle n° 2 (C2)

Info-sup S2
Epita

2 mars 2020 - 10 : 00

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- ☐ Durée : 2h00



Exercice 1 (Un peu de cours... – *4 points*)

Soit l'arbre général **A** représenté sous la forme binaire **premier fils - frère droit** :

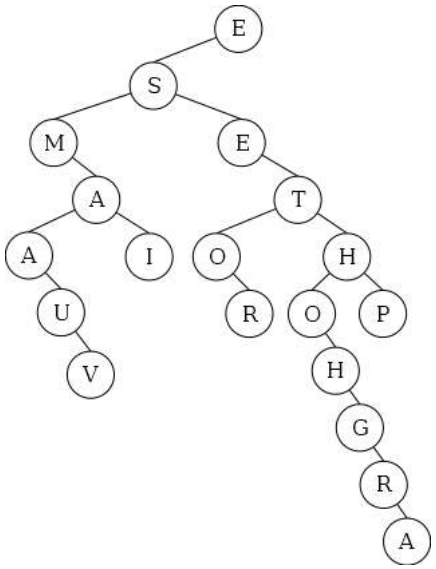


FIGURE 1 – Arbre général **A** sous forme binaire **premier fils - frère droit**

1. Quelle est la taille de l'arbre `A` ?
2. Quelle est la hauteur de l'arbre `A` ?
3. Quelle est la longueur de cheminement interne de l'arbre `A` ?
4. Quelle est la profondeur moyenne externe de l'arbre `A` ?
5. Donner la liste des nœuds de l'arbre `A` rencontrés en ordre suffixe.
6. Donner la liste des nœuds de l'arbre `A` rencontrés en ordre hiérarchique.

Exercice 2 (Carré magique – *4 points*)

Définition :

Un carré magique d'ordre n est composé de n^2 entiers, écrits sous la forme d'un tableau carré. Ces nombres sont disposés de sorte que leurs sommes sur chaque rangée, sur chaque colonne et sur chaque diagonale principale soient égales.

	0	1	2	3	4
0	11	18	25	2	9
1	10	12	19	21	3
2	4	6	13	20	22
3	23	5	7	14	16
4	17	24	(1)	8	15

FIGURE 2 – Carré magique d'ordre 5

Construction :

La *méthode Siamoise* permet de construire un carré magique d'*ordre n* impair contenant tous les entiers de 1 à n^2.

— La valeur 1 est placée sur la dernière ligne, au milieu ;
— Déplacement pour les valeurs suivantes :
  — si la valeur actuelle est un multiple de n, on "remonte" d'une case,
  — sinon, on descend d'une case vers la droite.

Écrire la fonction `Siamese(`n`)` qui construit un carré magique d'ordre n (entier supérieur à 2 impair).

Exercice 3 (Sous-liste – *5 points*)

Écrire la fonction `sub_line(`M`, `L`)` qui vérifie si la liste L est incluse dans une des lignes de la matrice M (supposée non vide).

1	1	10	10	7
10	0	9	3	8
3	1	4	7	5
0	8	1	1	1
4	5	1	8	9
12	11	7	8	1
6	2	10	9	8

FIGURE 3 – `Mat1`

Exemples d'applications sur la matrice de la figure 3 :

```
>>> sub_line(Mat1, [1, 10])
True
>>> sub_line(Mat1, [1, 10, 7])
False
>>> sub_line(Mat1,[4, 5, 1, 8, 9])
True
>>> sub_line(Mat1, [7, 8, 1])
True
```

Exercice 4 (Arbre partiellement ordonné – *3 points*)

Définition :

Un *arbre partiellement ordonné* est un arbre binaire étiqueté tel que la valeur contenue dans tout nœud est inférieure ou égale aux valeurs contenues dans les sous-arbres de ce nœud. On considérera un arbre vide comme partiellement ordonné.

Écrire la fonction `priority(`B`)` qui vérifie si l'arbre binaire B (dont les clés sont des entiers strictement positifs) est partiellement ordonné.

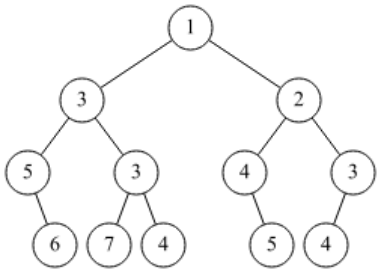


FIGURE 4 – `priority(B1)` → `True`

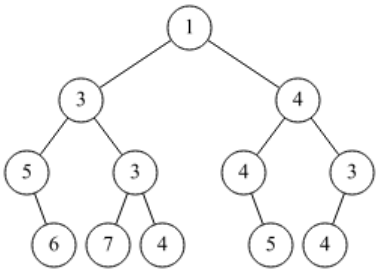


FIGURE 5 – `priority(B2)` → `False`

Exercice 5 (Largeur – *4 points*)

Définition :

Dans un arbre, la *largeur d'un niveau* est le nombre de nœuds de ce niveau.

La *largeur d'un arbre* est le maximum des largeurs de ses niveaux.

Écrire une fonction qui calcule la largeur d'un arbre binaire.

Exemple d'application sur l'arbre de la figure 4 :

```
>>> width(B1)
5
```

Annexes

Les arbres binaires

Les arbres binaires manipulés ici sont les mêmes qu'en td.
— `None` est l'arbre vide.
— L'arbre non vide est un objet de la class `BinTree` avec 3 attributs : `key`, `left`, `right`.

```
class BinTree:
    def __init__(self, key, left, right):
        self.key = key
        self.left = left
        self.right = right
```

Annexe : Fonctions et méthodes autorisées

Vous pouvez également utiliser la fonction `range`. Rappels :

```
>>> for i in range(10):
...     print(i, end=' ')
0 1 2 3 4 5 6 7 8 9

>>> for i in range(5, 10):
...     print(i, end=' ')
5 6 7 8 9
```

`List`

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes :

```
>>> help(list.append)
Help on method_descriptor:    append(...)
    L.append(object) -> None -- append object to end of L

>>> help(len)
Help on built-in function len in module builtins:    len(...)
    len(object)
    Return the number of items of a sequence or collection.
```

Matrices

Vous pouvez utiliser la fonction `initMat(`*line*, *col*, *val*`)` qui retourne une nouvelle matrice de *line* lignes sur *col* colonnes remplie de valeurs *val*.

Files

Les méthodes de la classe `Queue`, que l'on suppose importée :
— `Queue()` retourne une nouvelle file ;
— *q*`.enqueue(`*e*`)` enfile *e* dans *q* ;
— *q*`.dequeue()` supprime et retourne le premier élément de *q* ;
— *q*`.isempty()` teste si *q* est vide.

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.